Table of Contents

Filesystems

Using Access Control Lists for File Sharing

A common way to share files and/or directories with group members or others is to use the **chmod** command to change the permissions. However, **chmod** has limitations, so you may sometimes choose to use Access Control Lists (ACLs).

When you issue the command **chmod g+rx *filename***, for example, all the members in your group (**g**) gain read (**r**) and search/execute (**x**) access to that file, as shown below:

```
% ls -l filename
-rw------- 1 zsmith s0101 9 Jun 10 12:11 filename

% chmod g+rx filename

% ls -l filename
-rw-r-x--- 1 zsmith s0101 9 Jun 10 12:11 filename
```

However, **chmod** does not allow you to select which members of your group or which specific individuals outside of your group can access your files/directories. ACLs provide a mechanism for greater control of file sharing. There are two ACL commands:

setfacl

Set file access control lists.

```
SYNOPSIS
       setfacl [-bkndRLPvh] [{-m|-x} acl_spec] [{-M|-X} acl_file] file ...

       setfacl --restore=file
```

A detailed usage explanation of **setfacl** and its options can be found via **man setfacl**. Among the options listed:

1. The **-m** or **-M** option lets you "modify" the ACL, where **-m** expects an ACL on the command line and **-M** expects an ACL from a file or from standard input
2. The **-x** or **-X** option removes the ACL entries
3. The **-R** or **--recursive** option applies operations to all files and directories recursively
4. The **--test** option allows you to test the effect of changing the ACL without actually changing it
5. The **-b** option removes all extended ACL entries except the base entries of the owner, group, and others

getfacl

Get file access control lists.

```
SYNOPSIS
       getfacl [-dRLPvh] file ...

       getfacl [-dRLPvh] -
```

A detailed usage explanation of **getfacl** and its options can be found via **man getfacl**.

Note: Before you grant another user or group access to certain files or directories, make sure that access to the parent directory (where the files or directories reside) is also allowed.

Example 1

To allow another user (*jbrown*) to have read/execute (**rx**) permission on a file (*filename*) and to view the ACL before and after an ACL change:

```
% ls -l filename
-rw------- 1 zsmith s0101 9 Jun 10 12:11 filename

% getfacl filename
# file: filename
# owner: zsmith
# group: s0101
user::rw-
group::---
other::---

% setfacl -m u:jbrown:rx filename

% getfacl filename
# file: filename
# owner: zsmith
# group: s0101
user::rw-
user:jbrown:r-x
group::---
mask::r-x
other::---

% ls -l filename
-rw-r-x---+ 1 zsmith s0101 9 Jun 10 12:11 filename
```

Example 2

To remove all extended ACLs in Example 1 except the base entries of the owner, group, and others:

```
% setfacl -b filename

% ls -l filename
-rw------- 1 zsmith s0101 9 Jun 10 12:11 filename

% getfacl filename
# file: filename
# owner: zsmith
# group: s0101
user::rw-
group::---
other::---
```

Example 3

Continuing from Example 1, to test the granting of read/execute (**rx**) access to another group (group id 24176) without actually doing it:

```
% setfacl --test -m g:24176:rx filename
filename: u::rw-,u:jbrown:r-x,g::---,g:g24176:r-x,m::r-x,o::---,*

% getfacl filename
# file: filename
# owner: zsmith
# group: s0101
user::rw-
```

```
user:jbrown:r-x
group::---
mask::r-x
other::---
```

Example 4

To allow another user (*jbrown*) recursive access to a directory (**dir.abc** which contains a file *filename*):

```
% ls -ld dir.abc
drwx------ 2 zsmith s0101 17 Jun 10 13:19 dir.abc

% ls -l dir.abc
total 0
-rw------- 1 zsmith s0101 0 Jun 10 13:19 filename

% setfacl -R -m u:jbrown:rx dir.abc

% getfacl dir.abc
# file: dir.abc
# owner: zsmith
# group: s0101
user::rwx
user:jbrown:r-x
group::---
mask::r-x
other::---

% getfacl dir.abc/filename
# file: dir.abc/filename
# owner: zsmith
# group: s0101
user::rw-
user:jbrown:r-x
group::---
mask::r-x
other::---

% ls -ld dir.abc
drwxr-x---+ 2 zsmith s0101 17 Jun 10 13:19 dir.abc

% ls -l dir.abc
total 0
-rw-r-x---+ 1 zsmith s0101 0 Jun 10 13:19 filename
```

Example 5

Continuing from Example 4, to recursively remove all permissions user *jbrown* for a directory:

```
% setfacl -R -x u:jbrown dir.abc

% getfacl dir.abc
# file: dir.abc
# owner: zsmith
# group: s0101
user::rwx
group::---
mask::---
other::---

% getfacl dir.abc/filename
# file: dir.abc/filename
# owner: zsmith
# group: s0101
user::rw-
group::---
mask::---
other::---
```

For more information on ACLs, read **man acl**.

Using 'Giveto' to Copy Files and/or Directories to Another User

NAS's in-house developed **giveto** script is built on the use of Access Control Lists (ACLs). It allows one user (the giver) to copy files and/or directories to a **/nobackup** directory of another user (the recipient).

giveto is installed under **/usr/local/bin** on Pleiades and Lou.

In the example below, user zsmith gives a copy of his **dir.abc** directory on Pleiades to user jbrown. The steps describe the **giveto** command used by each of them, and the results.

1. User *jbrown* uses the command **giveto -i *zsmith*** to automatically (a) create an INCOMING directory (if it does not already exist) under her **/nobackup/*jbrown*** and (b) grant user *zsmith* read/write/execute permission on this directory.

```
pfe21:/u/jbrown% giveto -i zsmith
nobackup[1] =  /nobackup/jbrown

pfe21:/u/jbrown% ls -ld /nobackup/jbrown/INCOMING
drwxrwx---+ 2 jbrown s0202 4096 Jun 14 12:18 /nobackup/jbrown/INCOMING
```

2. User *zsmith* uses the command **giveto *jbrown* dir.abc** to automatically (a) create a subdirectory called ***zsmith*_0** under *jbrown*'s INCOMING directory, (b) copy **dir.abc** to **/nobackup/*jbrown*/INCOMING/*zsmith*_0**, (c) grant user jbrown read/write/execute permission on **/nobackup/*jbrown*/INCOMING/*zsmith*_0**, and (d) send an email to user *jbrown* regarding the copy.

```
pfe21:/home1/zsmith> ls -ld dir.abc
drwx------ 2 zsmith s0101 17 Jun 14 12:21 dir.abc/

pfe21:/home1/zsmith> giveto jbrown dir.abc
setfacl -m u:jbrown:rwx zsmith_0
setfacl -m u:jbrown:rwx zsmith_0/giveto.log
setfacl -m u:jbrown:rwx zsmith_0/dir.abc
setfacl -m u:jbrown:rwx zsmith_0/dir.abc/foo2
path =  /nobackup/jbrown/INCOMING/zsmith_0
total 12
drwxrwx---+ 3 zsmith  s0101 4096 Jun 14 12:29 .
drwxrwx---+ 3 jbrown  s0202 4096 Jun 14 12:29 ..
drwxrwx---+ 2 zsmith  s0101 4096 Jun 14 12:21 dir.abc
-rw-rwx---+ 1 zsmith  s0101   44 Jun 14 12:29 giveto.log
```

   Note: If the directory ***zsmith*_0** already exists prior to this step, ***zsmith*_1** would be used instead.

3. User *jbrown* receives an email from user *zsmith* with a subject line "giveto files". They see that the directory **dir.abc** has been copied successfully. Even though the directory **/nobackup/*jbrown*/INCOMING/*zsmith*_0** is still owned by user *zsmith*, user *jbrown* now has permission to read/write/execute files and directories under **/nobackup/*jbrown*/INCOMING/*zsmith*_0.**

```
pfe21:/u/jbrown% ls -lrt /nobackup/jbrown/INCOMING
total 4
drwxrwx---+ 3 zsmith s0101 4096 Jun 14 12:29 zsmith_0

pfe21:/u/jbrown%ls -lrt /nobackup/jbrown/INCOMING/zsmith_0
total 4
drwxrwx---+ 2 zsmith s0101 4096 Jun 14 12:21 dir.abc

pfe21:/u/jbrown%ls -lrt /nobackup/jbrown/INCOMING/zsmith_0/dir.abc
total 4
-rw-rwx---+ 1 zsmith s0101 8 Jun 14 12:21 foo2

pfe21:/u/jbrown%getfacl /nobackup/jbrown/INCOMING/zsmith_0/dir.abc
```

```
# file: /nobackup/jbrown/INCOMING/zsmith_0/dir.abc
# owner: zsmith
# group: s0101
user::rwx
user:jbrown:rwx
group::---
mask::rwx
other::---
```

Read **man giveto** for more information.

The **giveto** *script was created by NAS staff member Arthur Lazanoff.*

Quota Policy on Disk Space and Files

Filesystems on Pleiades and Lou have the following types of quotas:

- Limits on the total disk space occupied by your files
- Limits on the number of files (represented by inodes) you can store, regardless of size. For quota purposes, directories count as files.

Hard and Soft Quota Limits

NAS quotas have hard limits and soft limits. Hard limits should never be exceeded. Soft limits can be exceeded temporarily, for a grace period of 14 days. If your data remains over the soft limit for more than 14 days, the soft limit is enforced as a hard limit. On some filesystems, this means that write attempts will fail. On other filesystems, it will mean that your PBS jobs will no longer start. To reduce your data to below the quota limits, you can delete unneeded files or copy important files elsewhere, such as the Lou mass storage system, and then remove them locally.

Filesystem quotas are shown in the following table:

	Pleiades	**Lou**
$HOME	NFS	XFS
Space: soft	8 GB	none
Space: hard	10 GB	none
Inode: soft	none	250,000
Inode: hard	none	300,000
/nobackup	Lustre /nobackupp*X*	N/A
Space: soft	1 TB	N/A
Space: hard	2 TB	N/A
Inode: soft	500,000	N/A
Inode: hard	600,000	N/A

To learn how to check your disk space, inode usage, and quotas, see the following articles:

- Pleiades Home Filesystem
- Pleiades Lustre Filesystems
- The Lou Mass Storage System

Email Warnings and Consequences

It is expected that your data will exceed your soft limits as needed. When this occurs, you will begin to receive daily emails to inform you of your current disk space and how much of your grace period remains. However, if your data is still over the soft limit or if you reach the hard limit, restrictions are put in place.

The following table shows the quota enforcement policy for each type of filesystem.

Filesystem	**Grace Period**	**Enforcement**
/nobackupp*X*	2 weeks	Batch jobs won't run
/nobackupnfs2	2 weeks	Writes are disabled

Pleiades home directory	2 weeks	Writes are disabled
Lou home directory	2 weeks	Writes are disabled

More details are available in the following sections.

Lustre Filesystems (/nobackupp*X*)

If your grace period has expired, then your batch queue access is restricted and no new jobs can be run until your data on the Lustre filesystem is reduced to an amount below the soft limit. Any jobs that are running will continue to run. If you reach the hard limit, your batch access is immediately restricted, but any running jobs will continue.

Pleiades Home Directory and /nobackupnfs2

If your grace period has expired, then any writes to that filesystem will fail, and any jobs that attempt to write to the filesystem will fail. You will be unable to write to the filesystem until your data is reduced to an amount below the soft limit. Similarly, if you reach the hard limit, any writes will immediately fail and you will be unable to write to the filesystem until your data is reduced to an amount below the hard limit.

Lou Home Directory

If your grace period has expired, then any writes that attempt to create new files will fail. You will be unable to create any new files until the number of files is reduced to below the soft limit. If you reach the hard limit, any writes that attempt to create new files will fail until the number of files is reduced to below the hard limit.

The maximum size of a tar file moved to Lou should not exceed 2 TB. If you need to archive larger files, please contact the NAS Control Room at support@nas.nasa.gov for assistance.

Note: The Shift tool, which is a convenient way to transfer files to Lou, can created a set of smaller independent tar files from a single large directory. For more information, see Shift File Transfer Overview.

Changing Your Quotas

If an account needs larger quota limits, send an email justification to support@nas.nasa.gov. Your request will be reviewed by management for approval.

Pleiades Home Filesystem

Pleiades Home Filesystems

The home filesystems on Pleiades are HPE/SGI NEXIS 9000 filesystems that are NFS-mounted on all of the Pleiades front-end (PFE) nodes and compute nodes. Each NAS user is provided a home directory on the Pleiades home filesystems, which are the home filesystems for Pleiades, Aitken, Electra, and Endeavour. After you are granted an account, your home directory is set up automatically during your first login.

Your home directory has a quota of 8-10 GB of storage. For temporary storage of larger files, use the Lustre /nobackup filesystems. For long-term storage, use the Lou mass storage system.

Quota Limits and Policy

Disk space quota limits are enforced on the Pleiades home filesystems. By default, the soft limit is 8 GB and the hard limit is 10 GB. There are no inode limits on the home filesystem.

To check your quota and usage on your home filesystem, run the **`quota -v`** command as follows:

```
%quota -v
Disk quotas for user username (uid xxxx):
     Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
saturn-ib1-0:/mnt/home2
                7380152  8000000 10000000          190950       0       0
```

The NAS quota policy states that if you exceed the soft quota (the number listed under **`quota`** in the above sample output), an email will be sent to inform you of your current usage and how much of the grace period remains. It is expected that you will occasionally exceed your soft limit; however, after 14 days, any attempts to write to the filesystem will result in error. Any attempt to exceed the hard limit (the number listed under **`limit`** in the above sample output) will result in error.

If you believe that you have a long-term need for higher quota limits on the Pleiades home filesystem, send an email justification to support@nas.nasa.gov. Your request will be reviewed by the HECC Deputy Project Manager for approval.

Note: For temporary storage of larger files, or a large number of files, use your Lustre /nobackupp*X* directory. For normal long-term file storage, transfer your files to the Lou mass storage systems.

See also: Quota Policy on Disk Space and Files.

TIP: If you receive the following error message when logging in, you won't be able to run X applications. This error is usually due to exceeding your home filesystem quota. To avoid the error, decrease your disk usage.

`/usr/X11R6/bin/xauth: error in locking authority file /u/username/.Xauthority`

Backup Schedule

Files on the home filesystem are backed up daily.

Pleiades Lustre Filesystems

Pleiades Lustre Filesystems

Pleiades, Aitken, Electra, and Endeavour share several filesystems intended to provide working space for compute jobs. These filesystems, called "nobackup" (/nobackupp*X*), provide over 90 petabytes (PB) of disk space and serve many thousands of cores. The /nobackup filesystems are Lustre-based filesystems managed under Lustre software version 2.*x*.

Using the Lustre /nobackup Filesystems

As the name suggests, these "nobackup" filesystems are for temporary use, and are not backed up. Lustre can handle many large files, but you cannot use the Lustre filesystems for long-term storage. If you want to save your files, move them to Lou.

Each Lustre filesystem is shared among many users. To learn how to achieve good I/O performance for your applications and avoid impeding the I/O operations of other users, read the related articles listed at the bottom of the page.

Lustre filesystem configurations are summarized at the end of this article.

WARNING: Any files that are removed from Lustre /nobackup filesystems *cannot* be restored. You should store essential data on the Lou mass storage system (lfe[*5-8*]) or on other, more permanent storage.

Which Lustre Filesystem is Assigned to Me?

Once you are granted a NAS account, one of the Lustre filesystems will be assigned to you. To find out which one, run the **`ls`** command as follows:

```
pfe21% ls -l /nobackup/your_username
```

In the output, look for the filesystem name (**`nobackupp`*`X`***). For example, the following output shows that the user's assigned filesystem is /nobackupp17:

```
lrwxrwxrwx 1 root root 19 Sep 23  2021 /nobackup/your_username -> /nobackupp17/your_username
```

Default Quota and Policy on /nobackup Filesystems

Disk space and inode quotas are enforced on the /nobackup filesystems. The default soft and hard quota limits for inodes are 500,000 and 600,000, respectively. Quotas for the disk space are 1 terabyte and 2 terabytes, respectively. To check your disk space, inode usage, and quota on your /nobackup filesystem, run the **`lfs`** command as follows:

```
% lfs quota -h -u username /nobackup/username
Disk quotas for user username (uid nnnn):
        Filesystem      used       quota   limit   grace   files    quota   limit   grace
/nobackup/username        4k      1.024T  2.048T       -       1   500000  600000       -
```

It is expected that your data will occasionally exceed the soft limit. However, after a 14-day grace period, your access to the batch queues will be restricted and you will not be able to run new jobs until your data is reduced below the soft limit. If you reach the hard limit, your batch access will immediately be restricted, but any running jobs will continue.

If you anticipate a long-term need for higher quota limits, please send a justification via email to support@nas.nasa.gov. Your request will be reviewed by the HECC Deputy Project Manager for approval.

For more information, see Quota Policy on Disk Space and Files.

Note: When a Lustre filesystem is full, the jobs writing to it will hang. A Lustre error with code **-28** in the system log file indicates that the filesystem is full. NAS support staff will typically send emails to those using the most space, with a request to clean up their files.

Lustre Filesystem Configurations

The way your Lustre filesystem is configured determines how your files are striped. There are two types of configurations: Progressive File Layout (PFL), and the standard Lustre configuration. Most of the /nobackup filesystems have PFL configurations.

Progressive File Layout Configurations

Lustre's Progressive File Layout (PFL) feature enables filesystems to dynamically change the stripe count for files based on their size. This means that the files are dynamically striped, therefore, you should ***not*** set custom stripe size or stripe counts.

The Lustre PFL filesystems are /nobackupp[*10-29*]; among them, /nobackupp[*17-19, 27-29*] are equipped with a small number of solid state drives (SSDs) in addition to the hard disk drives (HDDs), while /nobackupp[*10-13,15-16*] only have HDDs.

The HDD configurations of /nobackupp[*10-29*] are listed in the table below with the abbreviation nbp*X.* P = petabytes; T = terabytes; M = megabytes.

Lustre PFL HDD Configurations

Filesystem	nbp10	nbp11	nbp12,13,15	nbp17,18	nbp19	nbp27,28	nbp29
# of OSTs	46	69	23	32	16	32	16
size/OST	70.7 T	70.7 T	70.7 T	565 T	565 T	565 T	565 T
Total Space	3.2 P	4.8 P	1.6 P	18.6 P	9.3 P	18.6 P	9.3 P

For /nobackupp[*10-13,15-16*], which have only HDDs, a common default PFL stripe setting is used as follows:

Lustre PFL Stripe Counts per File Size

File Size	0 - 10 MB	10 MB - 17 GB	17 - 68 GB	> 68 GB
Stripe Count	1	4	8	16

For /nobackupp[*17-19, 27-29*], different PFL configurations are set among the SSD and HDD pools. To learn more, see Progressive File Layout with SSD and HDD Pools.

Standard Configurations

The standard Lustre filesystems are /nobackupp1 and 2; in the table below, these are abbreviated as nbp*X.* P = petabytes; T = terabytes; M = megabytes.

Standard Lustre Configurations

Filesystem	nbp1	nbp2

# of OSTs	144	342
size/OST	43.6/57.2 T	43.6/71.2 T
Total Space	7.1 P	18 P
Default Stripe Size	1 M	1 M
Default Stripe Count	4	4

Lustre Basics

A Lustre filesystem is a high-performance shared filesystem for Linux clusters that is managed with Lustre software. It is highly scalable and can support many thousands of client nodes, petabytes of storage, and hundreds of gigabytes per second of I/O throughput. The NAS Lustre filesystems are named "/nobackupp*X*."

Each Lustre filesystem is actually a set of many small filesystems, which are referred to as object storage targets (OSTs). The Lustre software presents the OSTs as a single unified filesystem.

For more information about OSTs and other Lustre filesystem components, see Main Lustre Components.

Useful Lustre Commands

The examples in this section use /nobackupp*17* as an example of a specific Lustre filesystem.

Listing Disk Usage and Quotas

To display disk usage and limits on your /nobackup directory:

```
pfe21% lfs quota -h -u username /nobackupp17
```

or

```
pfe21% lfs quota -h -u username /nobackup/username
```

To display usage on each OST, add the -v option:

```
pfe21% lfs quota -h -v -u username /nobackup/username
```

Listing Space Usage

To list space usage per OST and MDT, in human-readable format, for all Lustre filesystems or for a specific one:

```
pfe21% lfs df -h
pfe21% lfs df -h /nobackupp17
```

Listing Inode Usage

To list inode usage for all filesystems or for a specific one:

```
pfe21% lfs df -i
pfe21% lfs df -i /nobackupp17
```

Listing OSTs

To list all the OSTs for the filesystem:

```
pfe21% lfs osts /nobackupp17
```

Viewing Striping Information

To view the striping information for a specific file or directory:

```
pfe21% lfs getstripe filename
pfe21% lfs getstripe -d directory_name
```

Note: Omitting the **-d** flag will display striping for all files in the directory.

File Striping

Files on the Lustre filesystems are striped automatically. This means they are transparently divided into chunks that are written or read simultaneously across a set of OSTs within the filesystem. The chunks are distributed among the OSTs using a method that ensures load balancing. Files larger than 100 gigabytes (GB) *must* be striped in order to avoid taking up too much space on any single OST, which might adversely affect the filesystem.

Benefits include:

- Striping allows one or more clients to read/write different parts of the same file at the same time, providing higher I/O bandwidth to the file because the bandwidth is aggregated over the multiple OSTs.
- Striping allows file sizes larger than the size of a single OST.

However, striping small files increases the number of objects in each file, resulting in increased overhead due to network operationsâ sometimes creating server contention. This is not likely to cause problems for files larger than 4 megabytes (MB), but in many cases, striping can cause noticeably slower read/write performance for files smaller than 4 MB.

Progressive File Layout

With the deployment of the new Lustre progressive file layout (PFL) feature, different stripe settings for different segments of a file can be configured in such a way that a small stripe count is used for the beginning segment and larger stripe counts are used for latter segments as the file grows. The newer Pleiades Lustre filesystems, /nobackupp[*10-29*], have all been configured with NAS-selected PFL settings.

Important: In most cases, you should rely on these defaults and not manually set file striping yourself. The information in the following sections are useful when using the older, non-PFL filesystems, /nobackupp[*1-2*].

To learn more about filesystems configured with PFL, see Progressive File Layout with SSD and HDD Pools.

Selecting a Stripe Count

The default stripe count on /nobackupp[*1-2*] filesystems is currently 4. The following examples describe circumstances where it is beneficial to change the stripe count to a different number:

- Your program reads a single large input file, where many nodes read or write different parts of the file at the same time. You should stripe this file adequately to prevent all the nodes from reading from the same OST at the same time. This will avoid creating a

bottleneck in which your processes try to read from a single set of disks.
- You have other large files. You should restripe large files adequately to keep capacity balanced across the OSTs and maintain consistent performance for all users.
- Your program waits while a large output file is written. You should stripe the large file so that it will write faster, in order to reduce the amount of time the processors are idle.
- Your program periodically writes several small files from each processor. It is not usually necessary to have a stripe count greater than 1 for small files, and they will be randomly distributed across the OSTs to maintain good performance and capacity balance in aggregate.

Setting Stripe Parameters

There are default stripe configurations for each Lustre filesystem. To check the existing stripe settings on your directory, use the **`lfs getstripe`** command as follows:

```
pfe21% lfs getstripe -d /nobackup/username
```

In most cases, the only tuning you need to do is to change the number of OSTs that your files are written to. To change the number of OSTs, use the **`lfs setstripe`** command as follows:

```
pfe21% lfs setstripe -c stripe_count dir|filename
```

Note: The stripe settings of an existing file cannot be changed. If you want to change the settings of a file, create a new file with the desired settings and copy the existing file to the newly created file.

Examples of Striping

Newly created files and directories inherit the stripe settings of their parent directories. You can take advantage of this feature by organizing your large and small files into separate directories, then setting a stripe count on the large-file directory so that all new files created in the directory will be automatically striped. For example, to create a directory called "restart" with a stripe count of 8, run:

```
pfe21% mkdir restart
pfe21% lfs setstripe -c 8 restart
```

You can "pre-create" a file as a zero-length striped file by running **`lfs setstripe`** as part of your job script or as part of the I/O routine in your program. Then, you can write to that file later. For example, to pre-create the file "big_dir.tar" with a stripe count of 20, and then add data from the large directory "big_dir," run:

```
pfe21% lfs setstripe -c 20 big_dir.tar
pfe21% tar cf big_dir.tar big_dir
```

Advanced Parameters

The following stripe parameters are rarely needed, but could potentially be useful, depending on your application I/O:

Stripe size: **`-S`**

Sets the size of the chunks in bytes. Use with **`k`**, **`m`**, or **`g`** to specify units of KB, MB, or GB, respectively (for example, **`-S 2m`**). The specified size must be a multiple of 65,536 bytes (64 KB). The default size is 1 MB for all Pleiades Lustre filesystems; specify **`0`** to use the

default.

Stripe offset: **-o**
: Sets the index of the OST where the first stripe is to be placed. The default is -1, which results in random selection. Using a non-default value is *not* recommended.

See the **lfs man page** for more options and information.

Main Lustre Components

Lustre filesystem components include:

Metadata Server (MDS)
: Service nodes that manage all metadata operations, such as assigning and tracking the names and storage locations of directories and files on the OSTs. There is 1 MDS per filesystem.

Object Storage Target (OST)
: Storage devices where users' file data is stored. The size of each OST varies from approximately 7 terabytes (TB) to approximately 22 TB, depending on the Lustre filesystem. The capacity of a Lustre filesystem is the sum of the sizes of all OSTs. There are multiple OSTs per filesystem.

Metadata Target (MDT)
: A storage device where the metadata (name, ownership, permissions and file type) are stored. There is 1 MDT per filesystem.

Object Storage Server (OSS)
: Service nodes that run the Lustre software stack, provide the actual I/O service and network request handling for the OSTs, and coordinate file locking with the MDS. Each OSS can serve multiple OSTs. The aggregate bandwidth of a Lustre filesystem can approach the sum of bandwidths provided by the OSSes. There can be 1 or multiple OSSes per filesystem.

Lustre Clients
: Compute nodes that mount the Lustre filesystem, and access/use data in the filesystem. There are commonly thousands of Lustre clients per filesystem.

Lustre Progressive File Layout (PFL) with SSD and HDD Pools

Hard disk drives (HDDs) and solid state drives (SSDs) are the two main storage options commonly deployed in data centers. With HDD, data are stored magnetically in spinning disks, and data retrieval is slow; therefore, HDD is practical only for data that does not need to be accessed frequently. SSDs are a newer technology where data is stored in integrated circuits, which dramatically reduces access time. Because SSDs are more expensive than HDDs, a hybrid solution provides a good compromise between price and performance.

Six of the NAS Lustre filesystems, /nobackupp[*17-19, 27-29*], are configured with Progressive File Layout (PFL) with HDDs and SSDs configured as different Lustre pools. This article provides information about these filesystems using /nobackupp17 as an example.

SSD and HDD Pools

To view a list of a filesystem's object storage targets (OSTs) showing whether they are in the HDD pool or the SSD pool, use the **`lfs pool_list`** command. The OSTs are shown in hexadecimal format:

```
pfe% lfs pool_list nbp17.hdd-pool
Pool: nbp17.hdd-pool
nbp17-OST0000_UUID
nbp17-OST0001_UUID
..
nbp17-OST001e_UUID
nbp17-OST001f_UUID

pfe% lfs pool_list nbp17.ssd-pool
Pool: nbp17.ssd-pool
nbp17-OST0064_UUID
nbp17-OST0065_UUID
...
nbp17-OST0082_UUID
nbp17-OST0083_UUID
```

The available SSD space in each filesystem is much smaller than the HDD space. In the example below, the **`lfs df`** command shows that the size of each /nobackupp17 HDD OST is 565.0 terabytes (TB), while each SSD OST is only 30.3 TB. The output also shows both the hexadecimal (far left) and decimal (far right) labels of each OST.

```
pfe% lfs df -h /nobackupp17 | grep OST
nbp17-OST0000_UUID        565.0T      322.7T      213.8T  61% /nobackupp17[OST:0]
nbp17-OST0001_UUID        565.0T      323.5T      213.0T  61% /nobackupp17[OST:1]
...
nbp17-OST001e_UUID        565.0T      322.9T      213.7T  61% /nobackupp17[OST:30]
nbp17-OST001f_UUID        565.0T      325.1T      211.4T  61% /nobackupp17[OST:31]
nbp17-OST0064_UUID         30.3T       23.5T        6.4T  79% /nobackupp17[OST:100]
nbp17-OST0065_UUID         30.3T       23.5T        6.4T  79% /nobackupp17[OST:101]
...
nbp17-OST0082_UUID         30.3T       23.5T        6.5T  79% /nobackupp17[OST:130]
nbp17-OST0083_UUID         30.3T       23.5T        6.5T  79% /nobackupp17[OST:131]
```

Configuration of the /nobackupp[*17-19, 27-29*] Filesystems

The following table summarizes the configurations of /nobackupp[*17-19, 27-29*].

Filesystem	**nbp17**	**nbp18**	**nbp19**	**nbp27**	**nbp28**	**nbp29**
# of OSTs in HDD	32 (OST: 0-31)	32 (OST: 0-31)	16 (OST: 0-15)	32 (OST: 0-31)	32 (OST: 0-31)	16 (OST: 0-15)

Size/OST in HDD	565.0 T	565.0 T	565.0 T	565.0 T	565.0 T	565.0 T
Total Space in HDD	18.6 P	18.6 P	9.3 P	18.6 P	18.6 P	9.3 P
# of OSTs in SSD	32 (OST: 100-131)	16 (OST: 100-115)	16 (OST: 100-115)	32 (OST: 100-131)	16 (OST: 100-115)	16 (OST: 100-115)
Size/OST in SSD	30.3 T	60.5 T	29.6 T	30.3 T	60.5 T	29.6 T
Total Space in SSD	~ 1 P	~ 1 P	~ 0.5 P	~ 1 P	~ 1 P	~ 0.5 P

Progressive File Layout (PFL)

The /nobackupp[*17-19, 27-29*] filesystems take advantage of the Lustre PFL feature, which increases the stripe count of each file in a step-wise manner as the file grows in size. Specifically, PFL is built using composite layouts described by a series of components. Each component covers a non-overlapping portion of the file and has its own stripe setting in size, count, index and pool.

Default PFL Configurations

For each filesystem, a NAS-created default layout provides reasonable performance for a variety of file I/O patternsâ and relieves users of the need to determine and set striping.

To find the default layout of each filesystem, use the **`lfs getstripe`** command as follows:

```
pfe% lfs getstripe -d /nobackupp17
lcm_layout_gen:    0
...
 lcme_extent.e_start: 0
 lcme_extent.e_end:   268435456
  stripe_count:  1    stripe_size: 16777216   pattern:   raid0   stripe_offset: -1  pool:  ssd-pool

 ...
 lcme_extent.e_start: 268435456
 lcme_extent.e_end:   5368709120
  stripe_count:  16   stripe_size: 16777216   pattern:   raid0   stripe_offset: -1  pool:  ssd-pool

 ...
 lcme_extent.e_start: 5368709120
 lcme_extent.e_end:   EOF
  stripe_count:  16   stripe_size: 16777216   pattern:   raid0   stripe_offset: -1  pool:  hdd-pool
```

The output shows that /nobackupp17 is configured with three components:

- The first component setting applies to the file segment from the beginning up to 268435456 B (256 MB), with a stripe count of 1 and stripe size of 16 MB; this component uses ssd-pool.
- The second component setting applies to the file segment from 256 MB to 5 GB with a stripe count of 16 and stripe size of 16 MB; this component uses ssd-pool.
- The last component setting applies to the file segment from 5 GB to the end of the file with a stripe count of 16 and stripe size of 16 MB; this component uses hdd-pool.

With this composite layout, I/O to and from the first 5 GB of files will benefit from the higher performance of SSDs.

Note: The default component layouts may be adjusted by NAS system administrators as needed.

Customized PFL Configurations

If you have experience using the PFL feature, and you want to have more control of your application's I/O, you can choose your own composite layouts using the **lfs setstripe** command with various options, such as **--component-end (or -E), --stripe-size (or -S), --stripe-count (or -c), --overstripe-count (or -C), --pool (or -p)**. For example:

```
lfs setstripe \
   -E 256M -C 1  -S 1M -p ssd-pool \
   -E 4G   -C 4  -S 1M -p ssd-pool \
   -E EOF  -C 16 -S 1M -p hdd-pool \
directory_name_or_filename
```

Note: If not explicitly set, files inherit layouts from their parent directory.

WARNING: SSD space is limited. Therefore, do not set striping for very large files to use only the SSD pool.

Mirroring and Migration of Data from SSD to HDD

As the limited SSD space fills up, data that is stored in SSDs is migrated over to HDDs to free up SSD space for I/O by new jobs. This automatic migration is accomplished through a mirroring mechanism.

After a file is closed, SSD data is queued to be mirrored to the HDD. The data will be in dual stateâ stored on both the SSD and the HDDâ until SSD usage reaches a certain point; then, the SSD copy will be removed to free up space. Typically, the oldest files are removed first when SSD usage reaches 80%, and other files are removed more aggressively when usage reaches 90%. The mirroring and migration settings may be adjusted by NAS system administrators as needed.

Note: Mirrored data that exists in both SSDs and HDDs are counted twice against your quota.

To find out if a file is in dual state or has been migrated completely to HDDs, use the **lfs getstripe filename** command, as shown in the next two examples.

Output Showing Data in Dual State

If the **lfs getstripe filename** output shows entries with different values for **lcme_mirror_id**, the data is in dual state, as illustrated in this example:

```
pfe% lfs getstripe file1
file1
  lcm_layout_gen:    5
  lcm_mirror_count:  2
  lcm_entry_count:   4
    lcme_id:             65537
    lcme_mirror_id:      1     <-------
    lcme_flags:          init,prefer
    lcme_extent.e_start: 0
    lcme_extent.e_end:   268435456
      lmm_stripe_count:  1
      lmm_stripe_size:   16777216
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 128
      lmm_pool:          ssd-pool
      lmm_objects:
      - 0: { l_ost_idx: 128, l_fid: [0x800000409:0x413e0a:0x0] }
```

```
    lcme_id:             65538
    lcme_mirror_id:      1   <--------
    lcme_flags:          init,prefer
    lcme_extent.e_start: 268435456
    lcme_extent.e_end:   5368709120
      lmm_stripe_count:  16
      lmm_stripe_size:   16777216
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 103
      lmm_pool:          ssd-pool
      lmm_objects:
      - 0: { l_ost_idx: 103, l_fid: [0x340000409:0x4305bb:0x0] }
      - 1: { l_ost_idx: 131, l_fid: [0x500000409:0x449f9f:0x0] }
      .....
      - 14: { l_ost_idx: 120, l_fid: [0x9c000040a:0x415384:0x0] }
      - 15: { l_ost_idx: 127, l_fid: [0x7c0000409:0x4100a6:0x0] }

    lcme_id:             65539
    lcme_mirror_id:      1  <---------
    lcme_flags:          init
    lcme_extent.e_start: 5368709120
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  16
      lmm_stripe_size:   16777216
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 27
      lmm_pool:          hdd-pool
      lmm_objects:
      - 0: { l_ost_idx: 27, l_fid: [0xfc0000400:0xd887b2:0x0] }
      - 1: { l_ost_idx: 26, l_fid: [0xb00000404:0xd8ed7c:0x0] }
      ....
      - 14: { l_ost_idx: 23, l_fid: [0xe40000400:0xd97afb:0x0] }
      - 15: { l_ost_idx: 9, l_fid: [0xe00000400:0x36d9a4:0x0] }

    lcme_id:             131073
    lcme_mirror_id:      2  <--------
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  16
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 9
      lmm_pool:          hdd-pool
      lmm_objects:
      - 0: { l_ost_idx: 9, l_fid: [0xe00000400:0x36d9bd:0x0] }
      - 1: { l_ost_idx: 19, l_fid: [0x120000040d:0xdaa1a0:0x0] }
      ....
      - 14: { l_ost_idx: 28, l_fid: [0xf40000400:0xd7c343:0x0] }
      - 15: { l_ost_idx: 11, l_fid: [0xc00000402:0x36d1eb:0x0] }
```

Output Showing Data Completely Migrated to HDDs

This example shows an older file that has been completely migrated to HDDs:

```
pfe% lfs getstripe file2
file2
  lcm_layout_gen:    2
  lcm_mirror_count:  1
  lcm_entry_count:   1
    lcme_id:             131073
    lcme_mirror_id:      2
    lcme_flags:          init
    lcme_extent.e_start: 0        <-----
    lcme_extent.e_end:   EOF      <-----
      lmm_stripe_count:  16
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 24
      lmm_pool:          hdd-pool
      lmm_objects:
```

```
      - 0: { l_ost_idx: 24, l_fid: [0xd8000040e:0x475b02:0x0] }
      - 1: { l_ost_idx: 3, l_fid: [0x130000040e:0x4733cd:0x0] }
      ....
      - 14: { l_ost_idx: 20, l_fid: [0xc8000040e:0x482bcc:0x0] }
      - 15: { l_ost_idx: 18, l_fid: [0xe8000040e:0x47cc62:0x0] }
```

Note: When your file is completely migrated to HDD (removed from SDD), the (**ctime)** of the file will change to the creation time of the mirrored copy on HDD, as described below.

Change Time (ctime) Example

When a file is mirrored, a second copy of the data is created. The file will retain both sets of data until you break the mirror, or when the filesystem automatically flushes the SSD poolâ at which time, the (**ctime**) will be updated to reflect the time the mirror was created.

In the following example, the **stat** command is used to show the time metadata of a test file that was created at 15:48:45:

```
% stat testfile
  File: 'testfile'
  Size: 59535           Blocks: 120        IO Block: 4194304 regular file
Device: 521e79ech/1377729004d   Inode: 342273918398214817  Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2022-02-14 15:48:45.000000000 -0800
Modify: 2022-02-14 15:48:45.000000000 -0800
Change: 2022-02-14 15:48:45.000000000 -0800
```

At time 15:50:27, the file is mirrored but the **ctime** remains unchanged:

```
% stat testfile
  File: 'testfile'
  Size: 59535           Blocks: 120        IO Block: 4194304 regular file
Device: 521e79ech/1377729004d   Inode: 342273918398214817  Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2022-02-14 15:48:45.000000000 -0800
Modify: 2022-02-14 15:48:45.000000000 -0800
Change: 2022-02-14 15:48:45.000000000 -0800
 Birth: -

pfe25 /nobackupp18/username % lfs getstripe testfile
testfile
  lcm_layout_gen:    1
  lcm_mirror_count:  2
  lcm_entry_count:   4
    lcme_id:             65537
    lcme_mirror_id:      1
    lcme_flags:          init,prefer
    lcme_extent.e_start: 0
    lcme_extent.e_end:   268435456
      lmm_stripe_count:  1
      lmm_stripe_size:   16777216
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 109
      lmm_pool:          ssd-pool
      lmm_objects:
      - 0: { l_ost_idx: 109, l_fid: [0x90000040a:0x53990:0x0] }

    lcme_id:             65538
    lcme_mirror_id:      1
    lcme_flags:          prefer
    lcme_extent.e_start: 268435456
    lcme_extent.e_end:   5368709120
      lmm_stripe_count:  -1
      lmm_stripe_size:   16777216
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: -1
      lmm_pool:          ssd-pool
```

```
    lcme_id:             65539
    lcme_mirror_id:      1
    lcme_flags:          0
    lcme_extent.e_start: 5368709120
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  16
      lmm_stripe_size:   16777216
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: -1
      lmm_pool:          hdd-pool

    lcme_id:             131073
    lcme_mirror_id:      2
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 0
      lmm_pool:          hdd-pool
      lmm_objects:
      - 0: { l_ost_idx: 0, l_fid: [0x2c0000409:0x30c3e:0x0] }
```

Later, the file is flushed from the SSD. At that time, the file's **`ctime`** is updated to reflect the creation time of the copy on HDD:

```
% stat testfile
  File: 'testfile'
  Size: 59535           Blocks: 120        IO Block: 4194304 regular file
Device: 521e79ech/1377729004d   Inode: 342273918398214817  Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2022-02-14 15:50:27.000000000 -0800
Modify: 2022-02-14 15:48:45.000000000 -0800
Change: 2022-02-14 15:50:27.000000000 -0800
 Birth: -
```

References

- Progressive File Layout
- File Level Redundancy
- Layout Enhancement High Level Design

Lustre Best Practices

At NAS, Lustre (/nobackup) filesystems are shared among many users and many application processes, which can cause contention for various Lustre resources. This article explains how Lustre I/O works, and provides best practices for improving application performance.

How Does Lustre I/O Work?

When a client (a compute node from your job) needs to create or access a file, the client queries the metadata server (MDS) and the metadata target (MDT) for the layout and location of the file's stripes. Once the file is opened and the client obtains the striping information, the MDS is no longer involved in the file I/O process. The client interacts directly with the object storage servers (OSSes) and object storage targets (OSTs) to perform I/O operations such as locking, disk allocation, storage, and retrieval.

If multiple clients try to read and write the same part of a file at the same time, the Lustre distributed lock manager enforces coherency so that all clients see consistent results.

Jobs being run on Pleiades contend for shared resources in NAS's Lustre filesystem. Each server that is part of a Lustre filesystem can only handle a limited number of I/O requests (read, write, stat, open, close, etc.) per second. An excessive number of such requests, from one or more users and one or more jobs, can lead to contention for storage resources.. Contention slows the performance of your applications and weakens the overall health of the Lustre filesystem. To reduce contention and improve performance, please apply the examples below to your compute jobs while working in our high-end computing environment.

Best Practices

Avoid Using ls -l

The **`ls -l`** command displays information such as ownership, permission, and size of all files and directories. The information on ownership and permission metadata is stored on the MDTs. However, the file size metadata is only available from the OSTs. So, the **`ls -l`** command issues RPCs to the MDS/MDT and OSSes/OSTs for every file/directory to be listed. RPC requests to the OSSes/OSTs are very costly and can take a long time to complete if there are many files and directories.

- Use **`ls`** by itself if you just want to see if a file exists
- Use **`ls -l`** ***`filename`*** if you want the long listing of a specific file

Avoid Having a Large Number of Files in a Single Directory

Opening a file keeps a lock on the parent directory. When many files in the same directory are to be opened, it creates contention. A better practice is to split a large number of files (in the thousands or more) into multiple subdirectories to minimize contention.

Avoid Accessing Small Files on Lustre Filesystems

Accessing small files on the Lustre filesystem is not efficient. When possible, keep them on an NFS-mounted filesystem (such as your home filesystem on Pleiades /u/*username*) or copy them

from Lustre to /tmp on each node at the beginning of the job, and then access them from /tmp.

Keep Copies of Your Source Code on the Pleiades Home Filesystem and/or Lou

Be aware that files under /nobackup are *not* backed up. Make sure that you save copies of your source codes, makefiles, and any other important files on your Pleiades home filesystem. If your Pleiades home directory quota isn't large enough to keep all of these files, you can request a larger quota and/or create tarred copies of these files on Lou.

Avoid Accessing Executables on Lustre Filesystems

There have been a few incidents on Pleiades where users' jobs encountered problems while accessing their executables on the /nobackup filesystem. The main issue is that the Lustre clients can become unmounted temporarily when there is a very high load on the Lustre filesystem. This can cause a bus error when a job tries to bring the next set of instructions from the inaccessible executable into memory.

Executables run slower when run from the Lustre filesystem. It is best to run executables from your home filesystem on Pleiades. On rare occasions, running executables from the Lustre filesystem can cause executables to be corrupted. Avoid copying new executables over existing ones of the same name within the Lustre filesystem. The copy causes a window of time (about 20 minutes) where the executable will not function. Instead, the executable should be accessed from your home filesystem during runtime.

Limit the Number of Processes Performing Parallel I/O

Given that the numbers of OSSes and OSTs on Pleiades are about a hundred or fewer, there will be contention if a large number of processes of an application are involved in parallel I/O. Instead of allowing all processes to do the I/O, choose just a few processes to do the work. For writes, these few processes should collect the data from other processes before the writes. For reads, these few processes should read the data and then broadcast the data to others.

Understand the Effect of Stripe Counts/Sizes for MPI Collective Writes

For programs that call MPI collective write functions, such as **`MPI_File_write_all`**, **`MPI_File_write_at_all`**, and **`MPI_File_write_ordered`**, it is important to understand the effect of stripe counts on performance.

Background

MPI I/O supports the concept of collective buffering. For some filesystems, when multiple MPI processes are writing to the same file in a coordinated manner, it is much more efficient for the different processes to send their writes to a subset of processes in order to do a smaller number of bigger writes. By default, with collective buffering, the write size is set to be the same as the stripe size of the file.

With Lustre filesystems, there are two main factors in the SGI MPT algorithm that chooses the number of MPI processes to do the writes: the stripe count and number of nodes. When the number of nodes is greater than the stripe count, the number of collective buffering processes is

the same as the stripe count. Otherwise, the number of collective buffering processes is the largest integer less than the number of nodes that evenly divides the stripe count. In addition, MPT chooses the first rank from the first *n* nodes to come up with *n* collective buffering processes.

Note: Intel MPI behaves similarly to SGI MPT on Lustre filesystems.

Enabling Collective Buffering Automatically

You can let each MPI implementation enable collective buffering for you, without any code changes.

SGI MPT automatically enables collective buffering for the collective write calls using the algorithm described above. This method requires no changes in the user code or in the **mpiexec** command line. For example, if the stripe count is 1, only rank 0 does the collective writes, which can result in poor performance. Therefore, experimenting with different stripe counts on the whole directory and/or individual files is strongly recommended.

Intel MPI also does collective buffering, similar to SGI MPT, when the I_MPI_EXTRA_FILESYSTEM and I_MPI_EXTRA_FILESYSTEM_LIST variables are set appropriately, as follows:

```
mpiexec.hydra  -env I_MPI_EXTRA_FILESYSTEM on \
        -env  I_MPI_EXTRA_FILESYSTEM_LIST lustre \
        -np xx a.out
```

Enabling Collective Buffering via Code Changes

In this method, you provide "hints" in the source code to inform MPI what to do with specific files. For example:

```
   call MPI_Info_create(info, status)
   call MPI_Info_set(info, "romio_cb_write", "enable", STATUS)
   call MPI_Info_set(info, "striping_unit", "1048576", STATUS)
   call MPI_Info_set(info, "striping_factor", "16", STATUS)
   ...
   call MPI_File_open(MPI_COMM_WORLD, file_name, MPI_MODE_WRONLY, info, unit, status)
```

Note: The hints are only advisory and may not be honored. For example, SGI MPT 2.12r26 honors these hints, but MPT 2.14r19 does not. Intel MPI 5.0x honors these hints when the I_MPI_EXTRA_FILESYSTEM and I_MPI_EXTRA_FILESYSTEM_LIST variables are set appropriately, as follows:

```
mpiexec.hydra  -env I_MPI_EXTRA_FILESYSTEM on \
        -env  I_MPI_EXTRA_FILESYSTEM_LIST lustre \
        -np xx a.out
```

Stripe Align I/O Requests to Minimize Contention

Stripe aligning means that the processes access files at offsets that correspond to stripe boundaries. This helps to minimize the number of OSTs a process must communicate for each I/O request. It also helps to decrease the probability that multiple processes accessing the same file communicate with the same OST at the same time.

One way to **stripe-align** a file is to make the stripe size the same as the amount of data in the write operations of the program.

Avoid Repetitive "stat" Operations

Some users have implemented logic in their scripts to test for the existence of certain files. Such tests generate "stat" requests to the Lustre server. When the testing becomes excessive, it creates a significant load on the filesystem. A workaround is to slow down the testing process by adding **sleep** in the logic. For example, the following user script tests the existence of the files WAIT and STOP to decide what to do next.

```
touch WAIT
 rm STOP

 while ( 0 <= 1  )
  if(-e WAIT) then
    mpiexec ...
    rm WAIT
  endif
  if(-e STOP) then
    exit
  endif
 end
```

When neither the WAIT nor STOP file exists, the loop ends up testing for their existence as quickly as possible (on the order of 5,000 times per second). Adding **sleep** inside the loop slows down the testing.

```
touch WAIT
 rm STOP

 while ( 0 <= 1  )
  if(-e WAIT) then
    mpiexec ...
    rm WAIT
  endif
  if(-e STOP) then
    exit
  endif
  sleep 15
 end
```

Avoid Having Multiple Processes Open the Same File(s) at the Same Time

On Lustre filesystems, if multiple processes try to open the same file(s), some processes will not able to find the file(s) and your job will fail.

The source code can be modified to call the sleep function between I/O operations. This will reduce the occurrence of multiple, simultaneous access attempts to the same file from different processes.

```
100  open(unit,file='filename',IOSTAT=ierr)
      if (ierr.ne.0) then
       ...
      call sleep(1)
      go to 100
      endif
```

When opening a read-only file in Fortran, use **ACTION='read'** instead of the default **ACTION='readwrite'**. The former will reduce contention by not locking the file.

```
open(unit,file='filename',ACTION='READ',IOSTAT=ierr)
```

Avoid Repetitive Open/Close Operations

Opening files and closing files incur overhead and repetitive open/close should be avoided.

If you intend to open the files for read only, make sure to use **`ACTION='READ'`** in the open statement. If possible, read the files once each and save the results, instead of reading the files repeatedly.

If you intend to write to a file many times during a run, open the file once at the beginning of the run. When all writes are done, close the file at the end of the run.

See Lustre Basics for more information.

Use the Soft Link to Refer to Your Lustre Directory

Your /nobackup directory is created on a specific Lustre filesystem, such as /nobackupp17 or /nobackupp18, but you can use a soft link to refer to the directory no matter which filesystem it is on:

```
/nobackup/your_username
```

By using the soft link, you can easily access your directory without needing to know the name of the underlying filesystem. Also, you will not need to change your scripts or re-create any symbolic links if a system administrator needs to migrate your data from one Lustre filesystem to another.

Preserve Corrupted Files for Investigation

When you notice a corrupted file in your /nobackup directory, it is important to preserve the file to allow NAS staff to investigate the cause of corruption. To prevent the file from being accidentally overwritten or deleted by your scripts, we recommend that you rename the corrupted file using:

```
pfe% mv filename filename.corrupted
```

Note: Do not use **`cp`** to create a new copy of the corrupted file.

Report the problem to NAS staff by sending an email to support@nas.nasa.gov. Include how, when, where the corrupted file was generated, and anything else that may help with the investigation.

Best Practices for Non-PFL Filesystems

Important: The tips in this section apply only to /nobackupp[*1-2*].

Use a Stripe Count of 1 for Directories with Many Small Files

Note: Skip this tip if you are using the PFL filesystems, nobackupp[*10-29*].

If you must keep small files on Lustre, be aware that **`stat`** operations are more efficient if each

small file resides in one OST. Create a directory to keep small files in, and set the stripe count to 1 so that only one OST will be needed for each file. This is useful when you extract source and header files (which are usually very small files) from a tarfile. Use the Lustre utility **lfs** to create a specific striping pattern, or find the striping pattern of existing files.

```
pfe21% mkdir dir_name
pfe21% lfs setstripe -c 1 dir_name
pfe21% cd dir_name
pfe21% tar -xf tar_file
```

If there are large files in the same directory tree, it may be better to allow them to stripe across more than one OST. You can create a new directory with a larger stripe count and copy the larger files to that directory. Note that moving files into that directory with the **mv** command will not change the stripe count of the files. Files must be *created in* or *copied* to a directory to inherit the stripe count properties of a directory.

```
pfe21% mkdir dir_count_4
pfe21% lfs setstripe -c 4 dir_count_4
pfe21% cp file_count_1 dir_count_4
```

If you have a directory with many small files (less than 100 MB) and a few very large files (greater than 1 GB), then it may be better to create a new subdirectory with a larger stripe count. Store just the large files and create symbolic links to the large files using the symlink command **ln**.

```
pfe21%  mkdir dir_name
pfe21%  lfs setstripe -c 16 dir_name
pfe21%  ln dir_name/large_file  large_file
```

Increase the Stripe Count for Parallel Access to the Same File

Note: Skip this tip if you are using the PFL filesystems, nobackupp[*10-29*].

The Lustre stripe count sets the number of OSTs the file will be written to. When multiple processes access blocks of data in the same large file in parallel, I/O performance may be improved by setting the stripe count to a larger value. However, if the stripe count is increased unnecessarily, the additional metadata overhead can degrade performance for small files.

By default, the stripe count is set to 4, which is a reasonable compromise for many workloads while still providing efficient metadata access (for example, to support the **ls -l** command). However, for large files, the stripe count should be increased to improve the aggregate I/O bandwidth by using more OSTs in parallel. In order to achieve load balance among the OSTs, we recommend using a value that is an integral factor of the number of processes performing the parallel I/O. For example, if your application has 64 processes performing the I/O, you could test performance with stripe counts of 8, 16, and 32.

TIP: To determine which number to start with, find the approximate square root of the size of the file in GB, and test performance with the stripe count set to the integral factor closest to that number. For example, for a file size of 300 GB the square root is approximately 17; if your application uses 64 processes, start performance testing with the stripe count set to 16.

Restripe Large Files

Note: Skip this tip if you are using the PFL filesystems, nobackupp[*10-29*].

If you have other large files, make sure they are adequately striped. You can use a minimum of one stripe per 100 GB (one stripe per 10 GB is recommended), up to a maximum stripe count of

120. If you plan to use the file as job input, consider adjusting the stripe count based on the number of parallel processes, as described in the previous section.

If you have files larger than 15 TB, please contact User Services for more guidelines specific to your use case.

We recommend using the **shiftc** tool to restripe your files. For example:

1. Run **ls -lh** to view the size of the file(s):

   ```
   % ls -lh data/large_file data/huge_file
   -rw-rw---- 1 zsmith  g1001 555G Apr 14 22:21 data/large_file
   -rw-rw---- 1 zsmith  g1001 3.2T Apr 14 22:21 data/huge_file
   ```

   When a file is less than 1,200 GB, simply use one stripe per 10 GB. For a larger file, you can specify a maximum stripe count of 120.
2. Use **shiftc** to copy the file to a new file with this number of stripes:

   ```
   % shiftc --stripe=10g large_file large_file.restripe
   % shiftc --stripe=120 huge_file huge_file.restripe
   ```
3. Verify that the file was successfully copied. You should receive an email report generated by the **shiftc** command, or you can run **shiftc --status**. In the email or status output, check that the state of the operation is "**done".**
4. Move the new file in place of the old file:

   ```
   % mv large_file.restripe large_file
   % mv huge_file.restripe huge_file
   ```

For more information, see Using Shift for Local Transfers and Tar Operations.

Stripe Files When Moving Them to a Lustre Filesystem

Note: Skip this tip if you are using the PFL filesystems, nobackupp[*10-29*].

When you copy large files onto the Lustre filesystems, such as from Lou or from remote systems, be sure to use a sufficiently increased stripe count. You can do this before you create the files by using the **lfs setstripe** command, or you can transfer the files using the **shiftc** tool, which automatically stripes the files.

Note: Use **mtar** or **shiftc** (instead of **tar**) when you create or extract tar files on Lustre.

See the following articles for more information:

- Lustre Basics - Setting Stripe Parameters
- Shift Command Options
- Using mtar to Create or Extract Tar Files on Lustre

Reporting Problems

If you report performance problems with a Lustre filesystem, please be sure to include the time, hostname, PBS job number, name of the filesystem, and the path of the directory or file that you are trying to access.Your report will help us correlate issues with recorded performance data to determine the cause of efficiency problems.

Pleiades BeeGFS Filesystem

Pleiades BeeGFS Filesystem

A solid state drive (SSD)-based BeeGFS filesystem, /nobackupp3, is in early stages of testing with a few users. Currently, it is only accessible from Pleiades front-end systems (PFEs) and compute nodes. The compute nodes on Electra or Aitken cannot access this filesystem.

The total space on /nobackupp3 is 192 terabytes (TB) with 16 stripes, each with a maximum of 12 TB. With its current configuration, you cannot set or change the stripe counts. Quota limits may be enabled in the future.

To view information about the filesystem, run:

```
% beegfs-ctl --listtargets --cfgFile=/etc/beegfs/beegfs-nbp3.conf  --longnodes --spaceinfo
% beegfs-ctl --listtargets --cfgFile=/etc/beegfs/beegfs-nbp3.conf --state --longnodes
```

WARNING: There is no failsafe mechanism for /nobackupp3. If one SSD fails, the data it contains cannot be recovered. Therefore, use Shift to copy any files you want to keep to your Lustre **/nobackup** filesystem (for temporary storage) or to Lou (for long-term storage). Be sure to properly stripe data copied to Lustre.

Using /nobackupp3

To use /nobackupp3 in a PBS job, you must ensure that it is mounted on all compute nodes before the job starts. To do so, include it as a needed site in your PBS script. For example, the following line mounts /nobackupp3, along with /home1 and /nobackupp8:

```
#PBS -l site=needed=/home1+/nobackupp3+/nobackupp8
```

Important: A job requesting Broadwell nodes might be run on Electra's Broadwell nodes instead of Pleiades. To run a job that uses Broadwell nodes and accesses /nobackupp3, you must specify Pleiades Broadwell nodes by using **#PBS -l site=static_broadwell**. For example:

```
#PBS -l select=xx:ncpus=28:mpiprocs=28:model=bro
#PBS -l site=static_broadwell
#PBS -l site=needed=/home1+/nobackupp3+/nobackupp8
```

For more information, see BeeGFS Basics.

BeeGFS Basics

A BeeGFS filesystem is a parallel cluster filesystem managed with the BeeGFS software. BeeGFS architecture is composed of four main services:

Management Service
: A very lightweight service for maintaining a list of all other BeeGFS services and their states.

Metadata Service
: Stores information about the data, such as directory information, file and directory ownership, and location of the user file contents on the storage targets. When a client opens a file, the metadata service provides the stripe pattern to the client. Metadata service is not involved in data read or write operations.

  There can be many server instances providing metadata services in a BeeGFS filesystem. Each service is responsible for its exclusive fraction of the global namespace. Having more metadata servers improves the overall system performance. Also, using faster processor cores for the metadata will have lower metadata access latency, providing better performance.

Storage Service
: Also referred to as the *object storage service*, this service stores striped user file contents.

  A storage server instance has one or more storage targets. The storage targets can be hard disk drives (HDDs) or the flash memory of solid state drives (SSDs).

  One strength of BeeGFS is its better performance compared to other filesystems in handling small I/O. This is because it automatically uses all available RAM on the storage servers for caching, so small I/O requests are aggregated into large blocks before the data is written to disk.

Client Service
: Mounts the filesystem to access the stored data. A BeeGFS client can be installed or updated without a system reboot.

Using the BeeGFS Command

The BeeGFS command **`beegfs-ctl`** can be used to perform administrative functions and show statistics:

```
pfe% which beegfs-ctl
/usr/bin/beegfs-ctl
```

To view **`beegfs-ctl`** options, run:

```
pfe% beegfs-ctl --help
```

BeeGFS Command Examples

To use these commands, you must specify a BeeGFS filesystem, using the **`--cfgFile`** option. In the examples, the Pleiades BeeGFS filesystem /nobackupp3 (**`nbp3`**) is specified.

- List the storage targets and their state:

  ```
  pfe% beegfs-ctl --listtargets --cfgFile=/etc/beegfs/beegfs-nbp3.conf --state --longnodes
  ```

- Find the total, free and % free space of the storage targets:

```
pfe% beegfs-ctl --listtargets --cfgFile=/etc/beegfs/beegfs-nbp3.conf  --longnodes --spaceinfo
```

- Find stripe pattern details of a directory or a file:

```
pfe% beegfs-ctl --getentryinfo --cfgFile=/etc/beegfs/beegfs-nbp3.conf /path/to/file/or/directory
```

- List the node(s) used as metadata server(s):

```
pfe% beegfs-ctl --listnodes --nodetype=meta --cfgFile=/etc/beegfs/beegfs-nbp3.conf
```

- List the nodes used as storage servers:

```
pfe% beegfs-ctl --listnodes --nodetype=storage --cfgFile=/etc/beegfs/beegfs-nbp3.conf
pfe% beegfs-ctl --listnodes --nodetype=storage --cfgFile=/etc/beegfs/beegfs-nbp3.conf --details
```

- List the storage pools:

```
pfe% beegfs-ctl --liststoragepools --cfgFile=/etc/beegfs/beegfs-nbp3.conf
```

Additional Resources

- Pleiades BeeGFS Filesystem
- Introduction to BeeGFS (PDF)